iOS 프로그래밍 Term 프로젝트 보고서

*// 텀 프로젝트는 1인 1프로젝트입니다.*

*// 모든 내용은 개조식으로 작성하지 말고* *서술식으로 작성하세요(존대말 사용금지)*

1. 학번 및 이름: 1971528 최유진
2. 제목: 고양이 백과사전 (Catopedia)
3. 앱 개요 및 구조

개요: 고양이 백과사전 어플을 통해 사용자는 고양이의 이름을 검색하거나 선호하는 성격 키워드를 선택하여 원하는 고양이 종에 대한 정보를 쉽게 얻을 수 있다. 또한, 사용자는 북마크 기능을 통해 관심 있는 고양이의 정보를 저장할 수 있다. 이는 고양이를 애호하는 모든 사람들이나 고양이를 입양하고자 하는 사람들에게 고양이에 대한 지식을 확장하도록 작용한다.

구조:

텍스트, 도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

1. 앱 기능(기술적 용어를 포함하나 일반 컴공전공 학생들을 이해할 수 있는 수준으로 기술)

Catopedia 화면 : 데이터셋의 모든 고양이 종 이름을 PickerView로 보여주고, Search Bar를 통해 입력한 고양이 이름이 PickerView에 나타나도록 하였다. 원하는 고양이를 선택하고 See Details 버튼을 클릭하면 선택한 고양이 정보를 가지고 상세 정보 화면으로 넘어간다.

Cat Match 화면 : 고양이 성격의 키워드들을 버튼으로 구현하였다. 원하는 키워드를 선택하고 Match 버튼을 클릭하면 선택된 키워드들로 데이터셋을 필터링하여 키워드에 해당하는 고양이 종들을 TableView에 나타나도록 하였다. TableViewCell를 클릭하면 해당 고양이 정보를 가지고 상세 정보 화면으로 넘어간다.

Bookmark 화면 : 북마크 버튼을 통해 Firebase Firestore에 저장된 고양이 종들을 CollectionView를 통해 보여준다. CollectionViewCell을 클릭하면 해당 고양이 정보를 가지고 상세 정보 화면으로 넘어간다.

상세 정보 화면 : ImageView, Label, TextView로 고양이 종의 상세 정보들을 구현하였다. 상단의 Bar Button Item으로 구현한 북마크 버튼을 클릭하면 해당 고양이 id가 Firebase Firestore에 저장된다.

1. 구현 내용(작성자와 담당 교수만 이해하면 될 정도의 심도 깊게 서술)

API 요청 : TheCatAPI를 사용하여 데이터셋을 확보하였다. 사용할 데이터들을 구조체 정의하였고, viewcontroller에서 비동기적으로 데이터 요청한다. 이미지의 경우, 데이터셋에 url주소로 존재해서 이미지 로드하는 작업을 백그라운드 스레드에서 비동기적으로 실행하였고 메인 스레드에서 이미지뷰에 이미지를 설정한다.

//CatData.swift

//API URL

**let** baseURL = "https://api.thecatapi.com/v1/breeds"

//API KEY

**let** apiKey = "live\_wLFZYYr0K2EzFic3c5VbSmdUvDoJMx3A04peNyX9gj4uDwM8J8GJxWPZ3X5p7m9N"

//고양이 데이터셋을 가져오는 함수

**func** fetchCatBreeds(completion: **@escaping** ([CatBreed]?, Error?) -> Void) {

**guard** **let** url = URL(string: baseURL) **else** { **return** } //URL 객체 생성

//HTTP 요청 객체 생성 및 설정

**var** request = URLRequest(url: url)

request.httpMethod = "GET"

request.addValue("application/json", forHTTPHeaderField: "Accept")

request.addValue(apiKey, forHTTPHeaderField: "x-api-key")

//CatBreedViewController.swift

//API로부터 고양이 데이터셋 받아옴

CatData().fetchCatBreeds { (breeds, error) **in**

**if** **let** error = error {

print("Error fetching cat breeds: \(error)")

} **else** **if** **let** breeds = breeds {

**self**.catBreeds = breeds

**self**.filteredCatBreeds = breeds //필터링 목록 초기화

**for** breed **in** breeds {

DispatchQueue.main.async {

**self**.catBreedPickerView.reloadAllComponents()

}

print("Breed ID: \(breed.id), Breed Name: \(breed.name), Description: \(breed.description), Life Span: \(breed.life\_span)")

}

}

}

//CatBreedDetailViewController.swift

//고양이 이미지를 로드하고 표시

**if** **let** imageUrlString = catBreed?.image?.url, //옵셔널 바인딩을 이용해서 imageUrl을 추출

**let** imageUrl = URL(string: imageUrlString) { //imageUrlString을 URL로 변환

DispatchQueue.global().async {

**if** **let** imageData = **try**? Data(contentsOf: imageUrl), //이미지 데이터를 로드

**let** image = UIImage(data: imageData) { //이미지 데이터를 UIImage로 변환

DispatchQueue.main.async { //메인 스레드에서 이미지 뷰에 이미지를 설정

**self**.catImageView.image = image

}

}

}

}

검색 결과 필터링 : UISearchBar를 통해 구현하였다. 입력한 데이터를 lowercased 메서드를 사용하여 대소문자 구분없이 검색하도록 하였고, contail 메서드를 사용하여 포함 여부를 확인한다. 필터링 후에 reloadAllComponents() 메서드를 호출하여 필터링 결과가 즉시 반영되도록 PickerView의 모든 컴포넌트를 다시 로드한다.

성격 키워드 필터링 : 키워드 버튼이 선택되어있다면 성격 데이터에 해당 키워드가 포함된 데이터만 찾도록 구현하였다. filterBreeds 함수는 필터링된 데이터를 담는 배열에 해당 특성들이 포함되는 품종만 남기고 필터링한다.

//각 버튼의 선택 상태에 따라 필터링

**if** affectionateButton.isSelected {

filteredCat = filteredCat.filter { $0.temperament.contains("Affectionate") }

}

**if** intelligentButton.isSelected {

filteredCat = filteredCat.filter { $0.temperament.contains("Intelligent") }

}

북마크 버튼 기능 : 북마크 버튼을 클릭하면 Firestore를 사용하여 "bookmarks" 컬렉션에 문서가 작성되도록 한다. 버튼이 이미 선택되었을 때 탭한다면 컬렉션에 해당 문서를 삭제하도록 하고 버튼의 선택 상태를 해제한다. 반면에 버튼이 선택되지 않은 상태라면 컬렉션에 새 문서를 추가하고 버튼의 선택 상태를 활성화한다.

북마크 화면 : firestore의 "bookmarks" 컬렉션에서 북마크된 고양이의 id와 일치하는 고양이를 찾아서 bookmarkedCats 배열에 추가한다. Collection View에 해당 배열의 데이터를 넣어서 UI를 구현하였다.

//Firestore에서 북마크된 고양이 정보를 로드하는 메소드

**func** loadBookmarkedCats() {

**self**.bookmarkedCats = [] //배열 초기화

db.collection("bookmarks").getDocuments { (querySnapshot, err) **in**

**if** **let** err = err {

print("Error getting documents: \(err)")

} **else** {

**for** document **in** querySnapshot!.documents {

**let** id = document.documentID

//북마크된 고양이의 id와 일치하는 고양이를 찾아서 bookmarkedCats 배열에 추가

**if** **let** cat = **self**.catBreeds.first(where: { $0.id == id }) {

**self**.bookmarkedCats.append(cat)

}

}

//collectionView 새로고침합니다.

DispatchQueue.main.async {

**self**.collectionView.reloadData()

}

}

}

}

상세 정보 화면으로 넘어가기 전에 대부분의 뷰컨트롤러에서는 세그웨이를 사용하여 선택한 데이터나, 필터링된 데이터를 상세 정보 화면 뷰 컨트롤러에 전달한다.

//세그웨이를 사용하여 필터링된 고양이 데이터 전달

**override** **func** prepare(for segue: UIStoryboardSegue, sender: **Any**?) {

**if** segue.identifier == "ShowMatchResult", //세그웨이 식별자

**let** resultViewController = segue.destination **as**? CatMatchResultViewController {

filterBreeds() //필터링 함수 호출

//필터링된 목록을 다음 뷰 컨트롤러에 전달합니다.

resultViewController.filteredCat = filteredCat

}

}

// 구현된 화면의 스크린 샷

텍스트, 스크린샷, 휴대 전화, 정보기기이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트, 휴대 전화, 정보기기이(가) 표시된 사진

자동 생성된 설명텍스트, 휴대 전화, 멀티미디어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 휴대 전화, 멀티미디어이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트, 멀티미디어, 휴대 전화이(가) 표시된 사진

자동 생성된 설명스크린샷, 휴대 전화, 통신 장치, 모바일 기기이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 휴대 전화, 멀티미디어이(가) 표시된 사진

자동 생성된 설명

고양이, 멀티미디어, 휴대 전화, 통신 장치이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 멀티미디어, 고양이이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 웹사이트, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

1. 결론**(반드시 작성)**

이번 프로젝트를 하면서 초기에 화면을 구상하고 스토리보드를 통해 UI를 구현한 후에 코드로 기능을 채워주는 과정이 재밌었습니다. 데이터 api를 찾아보고, 필터링을 구현하고, Collection View를 공부하며 구현한 후에 작동이 잘되는 화면을 보니 뿌듯함도 느꼈습니다. 다만, 필터링도 기존에는 좀 더 다양하게 되도록 구현하였으나 결과가 안 좋아서 삭제해야 했음이 아쉬웠고, scroll View나 라디오버튼을 통해 좀 더 완성도 있는 화면을 구현하고 싶었으나 여러 이유로 못하게 된 점이 아쉬웠습니다. 다음에 좋은 기회가 있다면 scroll View와 collection View를 공부하여 더 완성도 높은 화면을 구현해 보겠습니다.

1. 제출(elcass에 기한내 업로드)

* 보고서: project-보고서-학번-이름.[docx | hwp]
* 시연 동영상: project-동영상-학번-이름.mp4(발표한 학생은 제외)
* 소스코드: project-소스코드-학번-이름.zip(반드시 DerivedData 삭제)